Performance Concerns of Graph Algorithms
Andrew Lenharth
CS378 – Fall 2013

Primary Problem: Locality

Dense
- DS: Matrix, arrays
- Loops: over indexes
- Access pattern: affine formula
- Optimization: any safe reordering of the loop
  - Safety: dependence analysis

Graph
- DS: Pointers, arrays
- Loops: Over dynamic worklists
- Access pattern: indirect indexing
- Optimization: ??
  - Safety: ??

Cache Optimization – Size

- Fit more data in cache
- Optimal data structure depends on algorithm, needed data, and input
  - Optimal dense graph != sparse graph
  - Optimal for degrees <1,1,...,100> != <2...>

Cache Optimization – Partitioning

- Block data so nearby connect data uses same memory unit (L3, TLB, RAM)
- Process data within the same partition
- Defer partition crossing computations

Cache Optimization – Seq Access

- Maximize arrays iterated over in order
  - Generally the edge list for a node
  - Edge array v.s. edge list
- In conflict with mutability

The locality challenge
“Large memory footprint, low spatial and temporal locality impede performance”

Serial Performance of “approximate betweenness centrality” on a 2.67 GHz Intel Xeon 5560 (12 GB RAM, 8MB L3 cache)

Input: Synthetic R-MAT graphs (# of edges m = 8*n)

Problem Size (Log # of vertices)

Performance rate (Million Traversed Edges/s)

*Kamesh Madduri

*5X drop in performance
TLB Optimization – Hugepage

• Why store 4k per tlb entry when you can store 2M?

• Sandybridge L1 D-TLB
  – 64 4k entries or 32 2M entries
    • 256k or 64M of mapped data
  – 7 cycle miss latency

• If an average task accesses 32 nodes, randomly scattered through memory, how many expected TLB misses for each mapping size?
  – How does this affect average memory latency

Influencing Factors

• Density of input
• Distribution of input degree
• Required graph mutability
• Directionality
• Problem size
• Work per node
• Traversal pattern
• Structure of input data

Density

• Density of edges affects layout
  – Adj Matrix: O(n^2+sizeof(Edge Type))
  – Edge List: O(n sizeof(ptr) + m sizeof(ety + index + ptr))
  – Edge Array: O(n sizeof(ptr) + m sizeof(ety + index))
  – Operation costs vary

Degree distribution

• Some common graphs are very uneven
  – Inline low degree edge arrays
  
<table>
<thead>
<tr>
<th>Network</th>
<th>n</th>
<th>m</th>
<th>Max. out-degree</th>
<th>% of vertices w/ out-degree 0,1,2</th>
</tr>
</thead>
<tbody>
<tr>
<td>Orkut</td>
<td>3.07M</td>
<td>229M</td>
<td>32K</td>
<td>5</td>
</tr>
<tr>
<td>LiveJournal</td>
<td>5.28M</td>
<td>77.4M</td>
<td>9K</td>
<td>40</td>
</tr>
<tr>
<td>Flickr</td>
<td>1.86M</td>
<td>22.6M</td>
<td>26K</td>
<td>73</td>
</tr>
<tr>
<td>YouTube</td>
<td>1.15M</td>
<td>4.94M</td>
<td>28K</td>
<td>76</td>
</tr>
<tr>
<td>R-MAT</td>
<td>8M-64M</td>
<td>8n</td>
<td>n^0.4</td>
<td></td>
</tr>
</tbody>
</table>

Mutability

• Graph data structure determines ease of mutability

• Supporting mutable structure imposes high cost
  – E.g. Edge list v.s. edge array
  – Node list v.s. node array

• Supporting mutable data is easy, serially anyway

Directionality

• In a directed graph, edges are associated with nodes (only have to track them for a node)
  – In edges or out edges

• In an undirected graph, edges are shared
  – Harder to represent, takes additional data.
  – Similar to a directed graph where you know both in edges and out edges for a node
Problem Size

• Large graphs may need explicitly partitioning so algorithm can differentiate between in-partition nodes and out-partition nodes
  – Controls out-of partition loads
  – Other partitions may be on disk or another machine

Work per Node

• A memory bound traversal behaves differently than a computation bound algorithm
  – Mesh algorithms may do a lot of floating point, there is computation to mask latency
  – SSSP, for example, is limited by how fast you can do random access.

Traversal pattern

• Some algorithms visit all nodes in the graph once
• Some algorithms do a lot of work in one place
  – Temporal locality

Structure

• A planer graph is very amenable to partitioning
  – Or more generally any graph based on nodes embedded in space
• Graphs with high degree variation may benefit from multiple representations (and code)